IN THE CLAIMS:

Please amendment claims as follows:

1. A method for detecting computational errors in a system comprising a digital processor executing a program, the method comprising steps of:

separating the program into computation segments;

compiling source code for at least one of the segments to generate two code sections, one of which is functionally redundant with respect to the other; and

generating comparison code for comparing results produced by execution of the two code sections;

wherein the processor is configured to:

~~executing~~ execute each of the code sections in a different computational domain to generate respective results;

~~comparing~~ compare the respective results using the comparison code to detect said computational errors; and

~~executing~~ execute one of the code sections to alter further flow of execution of the program only if the respective results are identical.

2. (Original) The method of claim 1, wherein said computational domain comprises a time domain.

3. (Original) The method of claim 1, wherein the compiling step includes compiling the source code to schedule execution thereof so that a minimum number of processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections.

4. (Original) The method of claim 3, wherein said minimum number of processor clock cycles is predetermined as a function of statistical properties of duration of disruptive events causing said computational errors.

5. (Original) The method of claim 1, wherein said computational domain comprises a spatial domain.

6. (Original) The method of claim 1, wherein the compiling step includes compiling the source code such that each of the code sections is executed using separate resources of the processor.

7. (Original) The method of claim 6, wherein said resources comprise functional units and partitioned registers.

8. (Original) The method of claim 7, wherein the partitioned registers are used to effect the detection and repair of errors in the registers and paths to/from the registers.

9. (Original) The method of claim 6, wherein the partitioned registers are utilized by encoding register names from a first set of registers into instructions in a first one of the code sections and encoding register names from a second set of registers into instructions in the other one of the code sections.

10. (Original) The method of claim 1, wherein the respective results are compared by executing the comparison code in a different computational domain from the domain in which one of the code sections was executed.

11. (Original) The method of claim 1, wherein the compiler uses an explicit scheduling aspect of the processor's instruction set to ensure

that the two code sections are each executed by a different set of functional units.

12. (Original) The method of claim 1, including performing error handling if a discrepancy between the respective results is found.

13. (Original) The method of claim 12, wherein said error handling includes at least one function selected from the group consisting of re-execution, failing, and trapping to an error handling routine.

14. (Original) The method of claim 1, wherein each of the computation segments receives a set of inputs, performs at least one computation on the input values, and exposes a set of outputs to further computation.

15. (Original) The method of claim 1, including the step of optimizing one of the two code sections to execute via different registers and functional units than the other one of the code sections.

16. (Original) The method of claim 1, wherein the compiling step employs code reorganization to dynamically translate the source code into the two code sections.

17. (Original) The method of claim 1, wherein the step of compiling the source code is performed by incrementally translating the source code.

18. A system for detecting computational errors in a digital processor executing a program, the system comprising a compiler executing on a processor configured to:

separate the program into computation segments;

compile source code for at least one of the computation segments to generate output comprising two redundant code sections, each of which is configured to execute in a different computational domain; ~~and~~

generate comparison code for comparing respective results produced by execution of the two code sections; and

indicate that one of said computational errors has been detected when the respective results are different.

19. (Original) The system of claim 18, wherein the processor:

executes each of the code sections in a different computational domain to generate respective results for each of the code sections;

compares the respective results using the comparison code; and

performs error handling, if a discrepancy between the respective results is found.

20. (Original) The system of claim 19, wherein the respective results are compared by executing the comparison code in a different computational domain from the domain in which one of the code sections was executed.

21. (Original) The system of claim 19, wherein the error handling includes at least one function selected from the group consisting of re-execution, failing, and trapping to an error handling routine.

22. (Original) The system of claim 18, including an optimizer for modifying the output of the compiler to schedule execution of the redundant code sections so that a minimum number of clock cycles elapse between execution of a first one of the sections and execution of the other one of the code sections.

23. (Original) The system of claim 18, including an optimizer for configuring one of the redundant code sections to execute via different registers and functional units than the other one of the code sections.

24. (Original) The system of claim 18, wherein the source code is compiled to schedule execution of the redundant code sections so that a minimum number of processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections.

25. (Original) The system of claim 24, wherein said minimum number of processor clock cycles is predetermined as a function of statistical properties of the duration of disruptive events causing said computational errors.

26. (Original) The system of claim 18, wherein the compiler compiles the source code such that each of the code sections is executed using a different set of functional units and partitioned registers of the processor.

27. (Original) The system of claim 26, wherein the partitioned registers are used to effect the detection and repair of errors in the registers and paths to/from the registers.

28. (Original) The system of claim 26, wherein the partitioned registers are utilized by encoding register names from a first set of registers into instructions in a first one of the code sections and encoding register names from a second set of registers into instructions in the other one of the code sections.

29. (Original) The system of claim 18, wherein the compiler uses an explicit scheduling aspect of the processor's instruction set to ensure that the two code sections are not executed by the same functional units.

30. (Original) The system of claim 18, wherein the processor:

executes each of the code sections in a different computational domain to generate respective results for each of the code sections;

compares the respective results using the comparison code; and

executes one of the code sections to alter further flow of execution of the program only if the respective results are identical.

31. A system for detecting computational errors in a digital processor executing a program, the system comprising:

means for compiling source code for at least part of the program to generate two code sections, one of which is functionally redundant with respect to the other;

means for generating comparison code for comparing results produced by execution of the two code sections;

wherein each of the code sections is executed by the processor in a different computational domain to generate respective results;

processor means for comparing the respective results using the comparison code to detect said computational errors; and

processor means for performing error handling, if a discrepancy between the respective results is found.

32. (Original) The system of claim 31, wherein the source code is compiled to schedule execution thereof so that a minimum number of

processor clock cycles elapse between execution of a first one of the code sections and execution of the other one of the code sections.

33. (Original) The system of claim 31, wherein the source code is compiled such that each of the code sections is executed using a different set of functional units and partitioned registers of the processor.

34. Computer product code ~~A software product~~ in a system comprising instructions, stored on computer-readable storage media, wherein the instructions, when executed by a ~~computer~~ digital processor, perform steps for detecting computational errors in ~~a~~ the digital processor executing a program, comprising:

compiling source code for at least part of the program to generate two code sections, one of which is functionally redundant with respect to the other;

generating comparison code for comparing results produced by execution of the two code sections;

wherein the processor is configured to:

~~executing~~ execute each of the code sections in a different computational domain to generate respective results;

~~comparing~~ compare the respective results using the comparison code to detect said computational errors; and

~~performing~~ perform error handling, if a discrepancy between the respective results is found.

35. (Original) The software product of claim 34, wherein said computational domain comprises a time domain.

36. (Original) The software product of claim 34, wherein said computational domain comprises a spatial domain.